CS 3110 Project 2:

Selection Problem

Auraiporn Auksorn(aauksorn@cpp.edu)

Part 1: Date sets, Test strategies and Results

(1) Introduction

Problem

- Given a list of n numbers, find the kth smallest element in the list.

Algorithms

- The first algorithm (Algorithm 1) is to sort the list and then return the kth smallest element.
- The second algorithm (Algorithm 2) is to apply the procedure Partition used in Quicksort and implement it both iteratively and recursively.
- The third algorithm (Algorithm 3) is to apply the Partition algorithm with the mm rule.
    - For Algorithm 3, it first divides a list of n numbers into n/5 sets of 5 elements. Find a median in each set, then recursively use select to find the median of medians.

Input

- There is an array of size n, where numbers of n depending on the experiments.
- There is an array that defines the values of Kth elements where k = 1, n/4, n/2, 3n/4, and n, so there are 5 values of kth in total.
- (There are 5 data sets of the lists of random numbers of size n, and each list is used to test each value of Kth, so total of 5 lists for each experiment.)
- There is a finite value to determine how many times to run each algorithm according to the given size of a list and the Kth value. In this project, the number of times is initialized to have the value of 20 times for the first experiment and 40 times for the second experiment.

    (1.) Number of times: 20

    There are 5 arrays of n, where n = 10, 50, 100, 250, 500, 1000, 1250, 1500, 2000, 2250, 2500, 3000, 3250, and 3500.

    3500 is the maximum capacity of n for all kth values = 1, n/4, n/2, 3n/4, and n.

    (2.) Number of times: 40

    There are 5 arrays of n, where n = 10, 50, 100, 250, 500, 1000, 1250, 1500, 2000, 2250, 2500, and 2500.

    2500 is the maximum capacity of n for kth = n/2, 3n/4, and n.

    3000 is the maximum capacity of n for kth = 1 and n / 4.

Test strategies

- There are 2 test strategies:

    (1.) Generate 5 lists of random numbers in the range of 10,000 positive integers of size n where n = {10, 50, 100, 250, 500, 1000, 1250, 1500, 2000, 2250, 2500, 3000, 3250,3500}. Define the Kth value to implement the following Select-kth 1- Select-kth 4.

- Select-kth 1 is to implement Algorithm 1 using the O(n log n) Merge sort sorting method.
- Select-kth 2 will implement the Algorithm 2 using the partition procedure of Quicksort iteratively
- Select-kth 3 will implement the Algorithm 2 using the partition procedure of Quicksort recursively.
- Select-kth 4 is a recursive procedure to implement the Algorithm 3 with mm rule.

- Then, record the time taken to find one of the given Kth values which are initialized in an array of kth = {1, n/4, n/ 2, 3n/4, n}. For example, it will compile the given kth value = 1 with the same list of numbers on different 4 algorithms, then it will compile the given kth value = n/4, n/2, 3n/4, and n consecutively.
- Run each Kth value on 5 different lists of generated random numbers by different 4 algorithms of Select-kth for 20 times. Finally, compute the time taken by calculating the average time of the time taken to find the given kth value on each list of generated random numbers by each algorithm in each trial. In other words, there are 20 trials because we run each algorithm with the same set of input data for 20 times. Then, we compute the total time taken in trial# 1 to trail#20 divided by number of times.

(2.) Repeat the same steps and follow the same format as test strategies (1.) except the total values of size n will be different, and the number of times will also be different.

- The test strategies (2.) requires generating 5 lists of random numbers in the range of 10,000 positive integers of size n. Run the program for n = 10, 50, 100, 250, 500, 1000, 1250, 1500, 2000, 2250, 2500, 3000 (with k = 1, n/4, n/2, 3n/4, and n) for 40 number of times.

Output:

- The output indicates the elements in the list and shows kth smallest value. The screenshots of the output are an example of finding kth = n where the return value would be the maximum number in the list. It also prints out the total time to run the program by each algorithm.

```
Given Kth = n, then test with the same list of numbers of size 10 for 20 times.
The given unsorted list of numbers is:
3180 9047 6482 1413 5976 8444 5077 9035 7158 6217

Kth smallest element of each algorithm is:
Algorithm #1:               9047
Algorithm #2 Iteratively:   9047
Algorithm #2 Recursively:   9047
Algorithm #3:               9047

The given sorted list of numbers is:
1413 3180 5077 5976 6217 6482 7158 8444 9035 9047

Average time of Algorithm #1:               3175
Average time of Algorithm #2 Iteratively:   425
Average time of Algorithm #2 Recursively:   270
Average time of Algorithm #3:               995
```

The output of the program when running it for 20 times.

The output of the program when running it for 40 times:

```
Given Kth = n, then test with the same list of numbers of size 10 for 40 times.
The given unsorted list of numbers is:
6495 1599 1880 6853 3293 1991 1578 5983 3584 7812

Kth smallest element of each algorithm is:
Algorithm #1:               7812
Algorithm #2 Iteratively:   7812
Algorithm #2 Recursively:   7812
Algorithm #3:               7812

The given sorted list of numbers is:
1578 1599 1880 1991 3293 3584 5983 6495 6853 7812

Average time of Algorithm #1:               3162
Average time of Algorithm #2 Iteratively:   552
Average time of Algorithm #2 Recursively:   422
Average time of Algorithm #3:               1230
```

Description

The selection.java implements selection problem, given a list of n numbers, find the kth smallest element in the list, by 4 different algorithms such as Mergesort sorting method, the procedure Partition in Quicksort with recursive and iterative function, and the procedure Partition with mm rule. Then, record the time taken to find kth smallest element by each algorithm for the analysis part.

(2) Main Components

Selection.java

Contains the following main components of the selection problem.

2.1 The code component to implement Algorithm 1 using the O(n log n) Mergesort sorting method and return the kth smallest element.

2.2 The code component to implement the second algorithm (Algorithm 2) both iteratively and recursively which applies the procedure Partition used in Quicksort.

2.3 The code component to implement the third algorithm (Algorithm 3) which applies the Partition algorithm with the mm rule

2.3 The code component to generate the list of random numbers to test 4 different algorithms

2.3 The code component in the record time function to perform the algorithm analysis by calculating the total time spent which is then divided by the number of times the selection is performed to obtain the time taken to solve the given instance.

(3) Results

- Select-kth 4 becomes faster than Select-kth 1 when there are less elements in a list of size n, but Select-kth 4 will be slower than Select-kth 1 as the size keeps growing. I think in theory Select-kth 4 is faster than Select-kth 1; however, in the implementation Select-kth 4 is slower than Select-kth 1.
- Overall, of the runtime result of kth smallest element and 4 different algorithms indicates that Select-kth 2 is always faster than Select-kth 3. Even though there are some scenarios where Select-kth 2 is slower than Select-kth 3, the differences of their runtime are very small. There are also some scenarios where Select-kth 2 and Select-kth 3 are about the same.
- Select-kth 2 and Select-kth 3 will take longest when finding the first smallest element (kth = 1), and when finding kth = n, the runtime of Select-kth 3 will becomes faster than the runtime of Select-kth 2.
- Select-kth 1, Select-kth 2, Select-kth 3, and Select-kth 4 algorithms are fastest when finding smallest kth = n.
- Select-kth 4 algorithm seems to the take the longest time to find any kth smallest elements.

Part 2: Theoretical Complexity Comparisons

According to the lecture of divide-and-conquer points out the theoretical complexity comparisons of the three algorithms, which are the followings:

The table shows the theoretical complexity comparisons of the three algorithms.

Algorithm	**Complexity**	
	Best Case	**Worst Case**
Algorithm #1: (Use sorting strategy like MergeSort, sort the whole list and then return the kth element of the sorted list.)	O(n Log n)	O(n Log n)
Algorithm #2: (Use "partition" (from Quick Sort) procedure repeatedly until we find the kth smallest element.)	O(n)	O(n^2) In case of k = 1 and k = n. (If the pivot is always the minimal or maximal element.)
Algorithm #3: (Same as Algorithm#2 except it always find a special element of the list called MM (Median of Medians) and use MM as the pivot.)	O(n)	O(n)

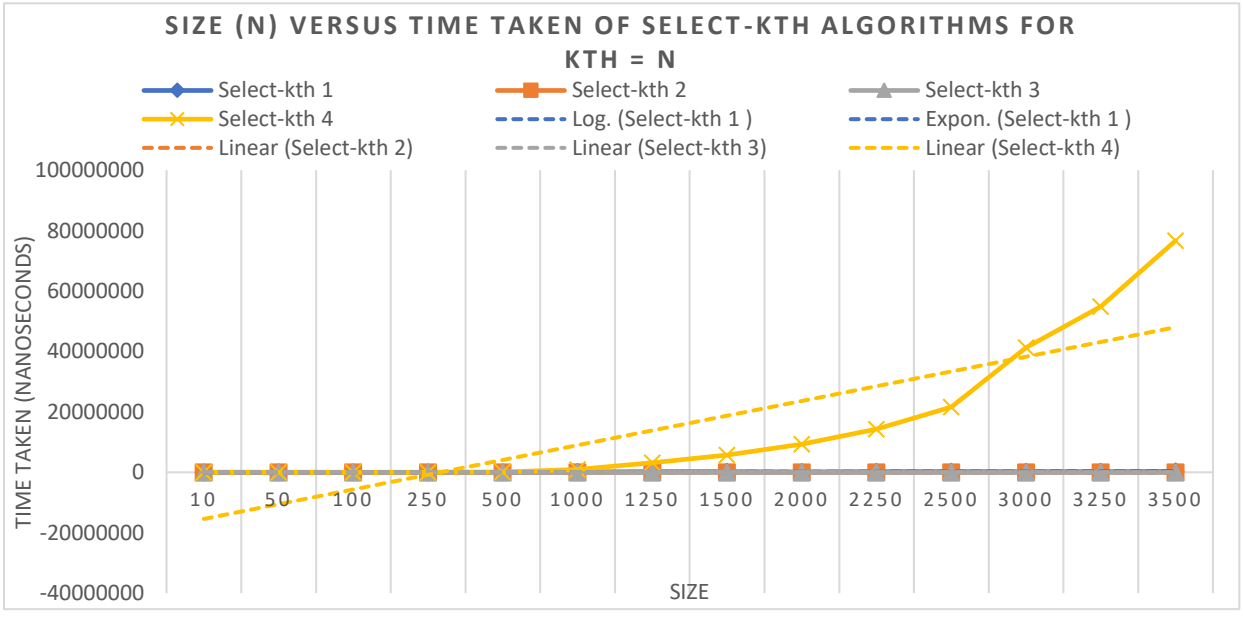
According to theoretical complexity, Algorithm #3 would be the most efficient since its worst case and its best case is O(n). Algorithm #2 its best case would also be faster than Algorithm #1, even though its worst case of Algorithm #2 is less efficient than Algorithm #1. Therefore, the analysis of this project will compare the performance and the time complexity of these three algorithms.

The First Test Strategies and the Second Test Strategies:

Evaluation of the results:

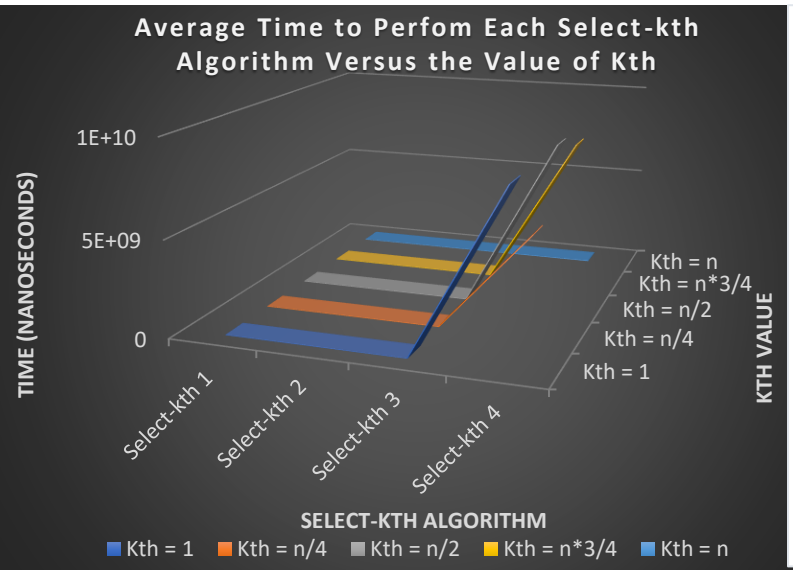
Since I have collected the data with two different test strategies; the first data set is running the program for 20 times, and the second data set is running the program for 40 times.

The graph shows the relationship of the time taken to find kth = n by different 4 select – kth algorithms versus the size of given list n, where n is 10, 50, 100, …, 3500

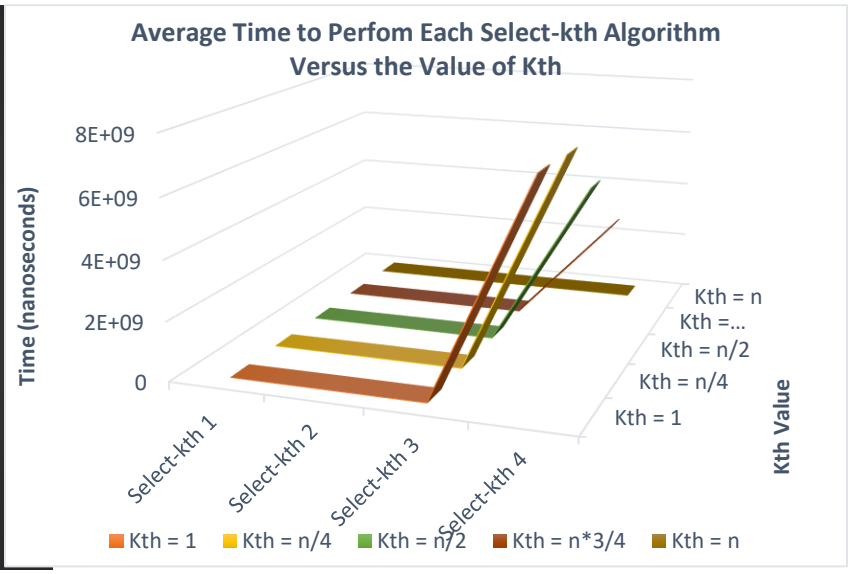


The result of 3 algorithms, Select-kth 1 (Merge Sort), Select-kth 2 (Quick Select iteratively), and Select-kth 3 (Quick Select Recursively), comes out to be linear. The line of Select-kth 4 algorithm tends to keep increasing and growing as the size becomes bigger. I think the reason that the line of Select-kth 4 algorithm behaves differently because it is the algorithm that will take longest time to solve kth select problem.

The graph indicates the average time of each Select-kth algorithm versus the kth value.



The black graph is the data set **from the first test strategies.**



The white graph is the data set **from the second test strategies**

The average time of this graph is obtained by the sum of time taken of each algorithm to find each kth value of all size n divided by the total numbers of size n = {10, 50, 100, 250, 500, 1000, 1250, 1500, 2250, 2500, 3000, 3250, 3500}, which there are 14 sets of data that used to test each kth value for the first test strategies. For the second test strategies the maximum of the size n is 2500 for kth = n/2, 3n/4, and n; for kth = 1 and n/4, the maximum of size n is 3000.

According to the observations, all Select-kth algorithms work perfectly fine when kth has the value of 1, n/4, n/2, and 3n/4. The graph function of these algorithms is the linear line. However, when the Median of Medians algorithm performed the select kth problem, the line tends to increase significantly. It is because the Select-kth 4 which is to apply the mm rule takes the longest time to accomplish the assigned task as it implemented 2 recursion functions.

The table shows the average time to perform each algorithm for each select-kth problem of all size n from the first test strategies.

Kth value	Select-kth 1	Select-kth 2	Select-kth 3	Select-kth 4
Kth = 1	153807.3571	1776587.071	1776768.143	8719935887
Kth = n/4	135959	1470219.214	1481193.571	5231735961
Kth = n/2	149147.0714	1343332.571	1365020.714	8737326060
Kth = n*3/4	128630.5714	841312.8571	838570.1429	7879586163
Kth = n	125211.7857	2915.571429	1987.142857	16299840.57

This table shows which kth value is the fastest or slowest; for instance, kth = 1, Select-kth 4 takes longest, and Select-kth 1 is the fastest. Also, Select-kth 2 and Select-kth 3 take about the same time, but Select-kth 2 is faster than Select-kth 3. When the value of kth equals to 1, Select-kth 2 and Select-kth 3 takes the longest time because kth = 1 is the worst case of quick select algorithm.

The table shows the average time to perform each algorithm for each select-kth problem of all size n from the second test strategies.

Kth value	Select-kth 1	Select-kth 2	Select-kth 3	Select-kth 4
Kth = 1	252368	10673178.08	10678786.75	7401364610
Kth = n/4	234725	10750598.67	10676151.83	7310563828
Kth = n/2	222168.3636	7722863.727	7921130.364	5570209641
Kth = n*3/4	227319.4545	4222082.455	4156160	3463386179
Kth = n	191353.1818	12103.63636	11324.36364	9842176.636

The result of the table from the second test points out the same pattern of data such as Select-kth 3 is fastest when kth = n, and Select-kth 4 is slowest when kth = 1. However, the key finding from this table shows that the numbers in this table are more precise and gives a better understanding for algorithm analysis part.

Part 3: Select-kth 2 Versus Select-kth 3

- **First Experiment:**
    - According to my observations from the tables of the first experiments, Select-kth 2 would probably always be faster than Select-kth 3 even though there are some results of time taken to find kth value that points out Select-kth 2 sometimes is slower than Select-kth 3.
    - Example of the scenarios where Select-kth 2 is not faster than Select-kth 3:
        - Finding kth = 1 at size 50, 1000, 1250, 1500, 2250, 2500, 3000, 3250.
        - Finding kth = n/4 at size 50, 100, 250, 500,1000, 2000, 2500, 3500
        - Finding kth = n/2 at size 50, 250, 500, 1000, 1250, 2250, 2500.
        - Finding kth = 3n/4 at size 50, 250, 2500, 3000, 3250.
        - Finding kth = n for all size of n Select-kth 2 is slower than Select-kth 3.

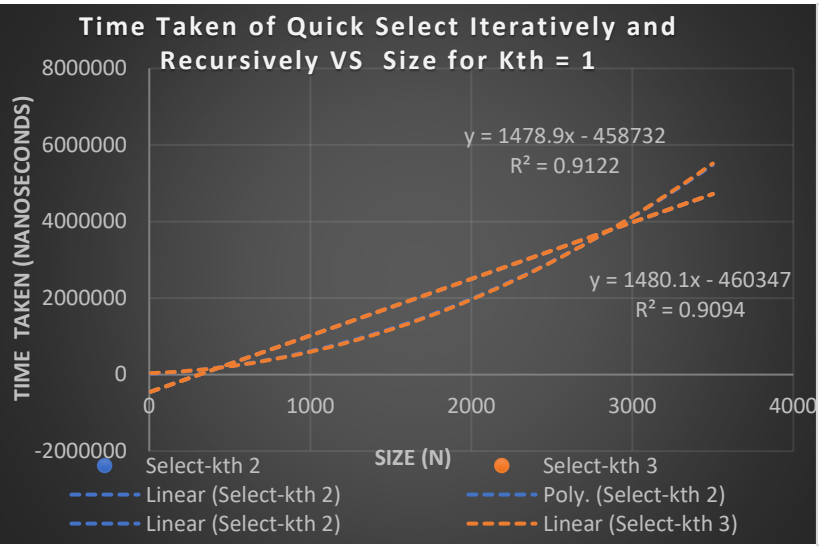
    Select-kth 2 is slightly slower than Select-kth 3. However, the difference of time taken in Select-kth 2 and Select-kth 3 is slightly different. Some of the time taken is almost the same.
- **Second Experiment:**
    - According to the tables of the second experiment when running the program for 40 times points out that Select-kth is always faster than Select-kth 3. Only in a few scenarios where Select-kth 3 is faster than Select-kth2.
    - Example of the scenarios where Select-kth 2 is not faster than Select-kth 3:
        - Finding kth = 1 at size 1250, 1500, 2000.
        - Finding kth = n/4 at size 100, 1250, 2000, 2500
        - Finding kth = n/2 at size 100, 250, 1000, 1250
        - Finding kth = 3n/4 at size 10, 1000, 1250, 1500, 2000, 2250
        - Finding kth = n for all size of n Select-kth 2 is slower than Select-kth 3.

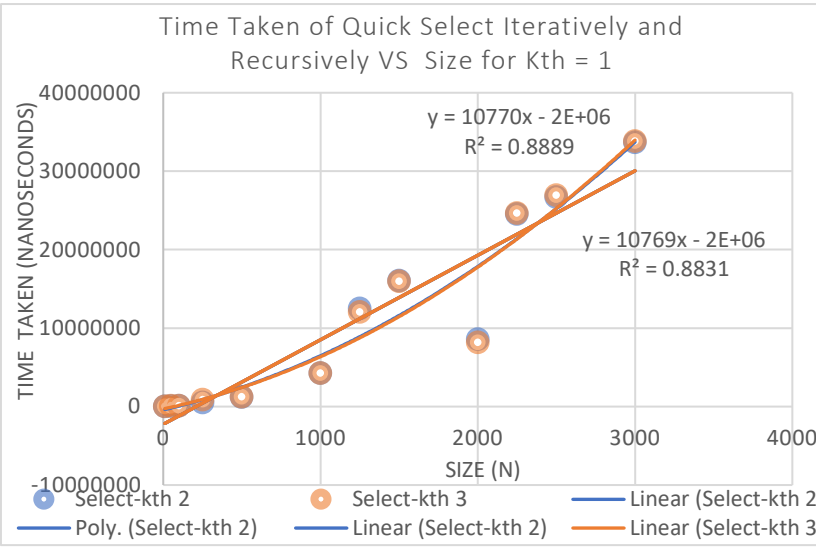
I think the greater number of times I run the program; I would get more accurate result. As the example above shows that when running the program for 40 times, there are less numbers of size n where Select-kth 2 is slower than Select-kth3 compared to 20 number of times to run the program. However, when finding the kth = n, shows that Select-kth 2 is not always faster than Select-kth 3 in the first and the second test strategies.

Note: The black graph is the data set **from the first test strategies.**

The white graph is the data set **from the second strategies**

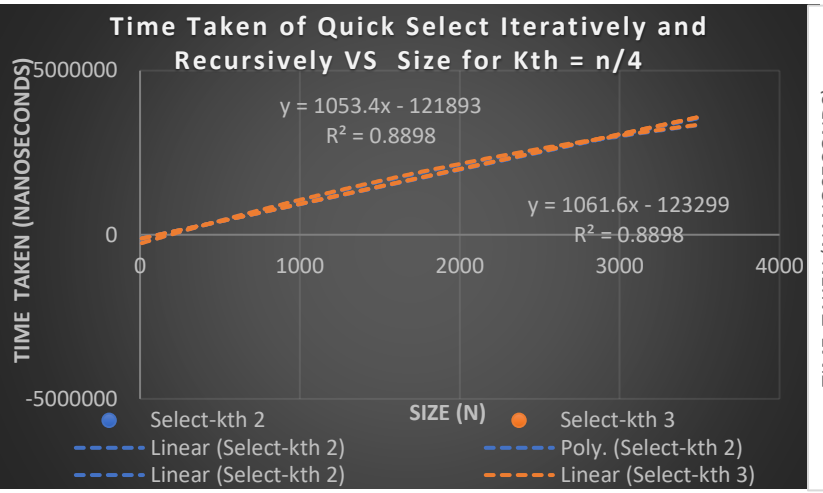
The graph shows the size versus the total time of algorithm Select-kth 2 and Select-kth 3 for a given value of kth equals to 1.

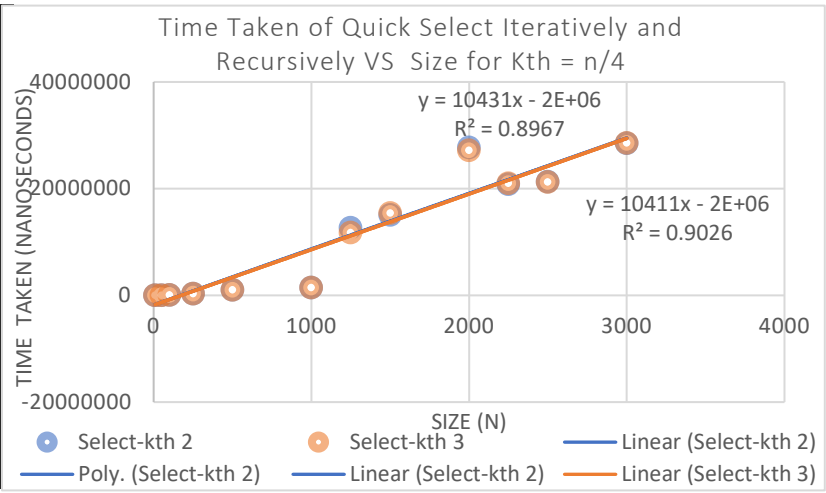




According to these two graphs show that the runtime is linear, but the graph of the second test provides more organized runtime result that lies on a linear trendline. The runtime from the first test of size n = 1000, 1250, and 2000 lies on the trendline of polynomial to the power of 2.
I think that might be Select-kth 2 and Select-kth 3 algorithms in the worst-case scenario.
Finally, from these two graphs I learned that when kth value = 1, the Select-kth 2 is faster than Select-kth 3 even though their runtime is slightly different. The graph of second test shows that the runtime of two algorithms is linear as we can see that the data points lie on the linear trendline.

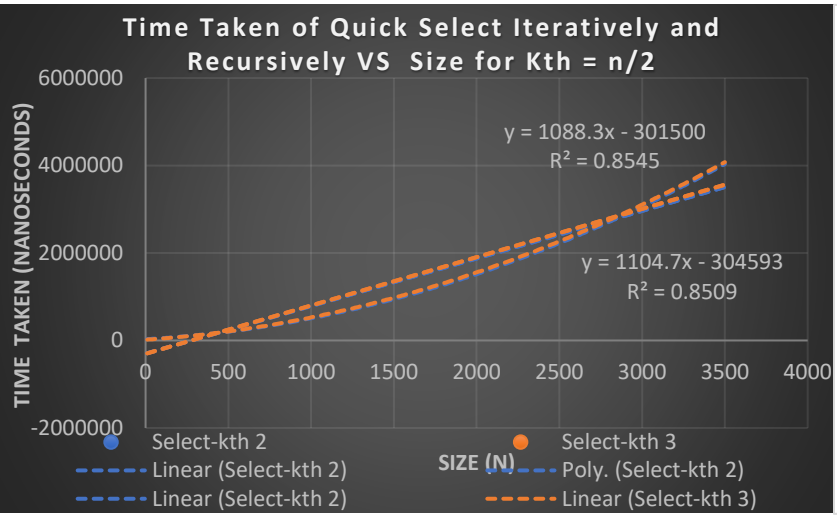
The graph shows the size versus the total time of algorithm Select-kth 2 and Select-kth 3 for a given value of kth equals to n/4.

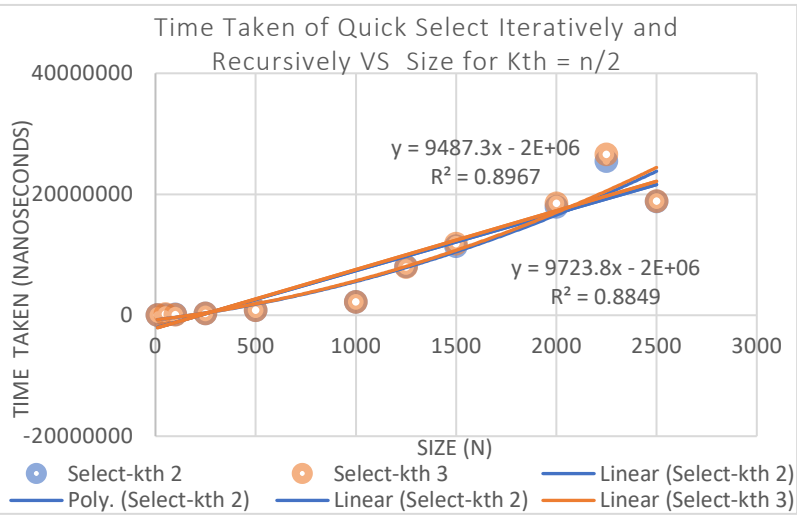




For these two graphs, I notice that the linear and polynomial trend line are about the same in the graph of the first test, and they are the same in the graph of the second test. There are more data points that lie on the trend line in the second test than the first test, so it confirms that the more number of times running the program; I obtained more accurate results when running the program for 40 times. These two graphs show that Select-kth 2 and Select-kth 3 have the time complexity of O(n). Even though the graph from the second test indicates that when kth = n/4, Select-kth 3 is faster than Select-kth 2, the graph from the first test confirms that Select-kth 2 is faster than Select-kth 3.

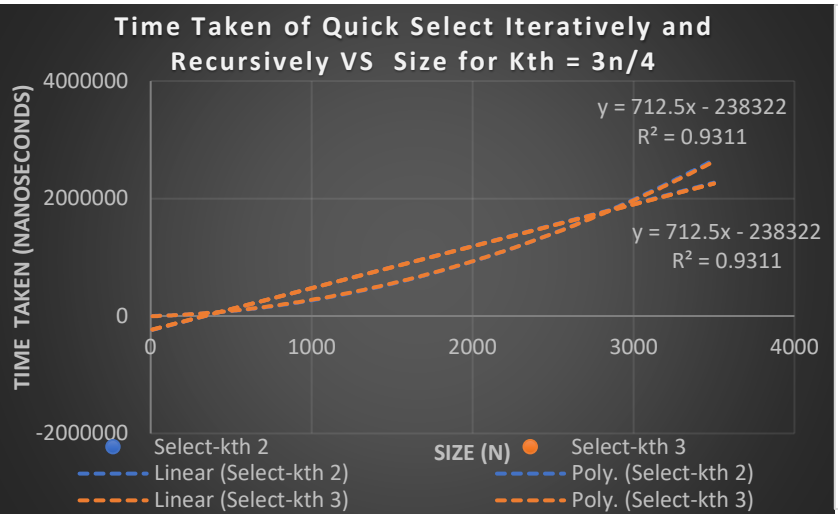
The graph shows the size versus the total time of algorithm Select-kth 2 and Select-kth 3 for a given value of kth equals to n/2.

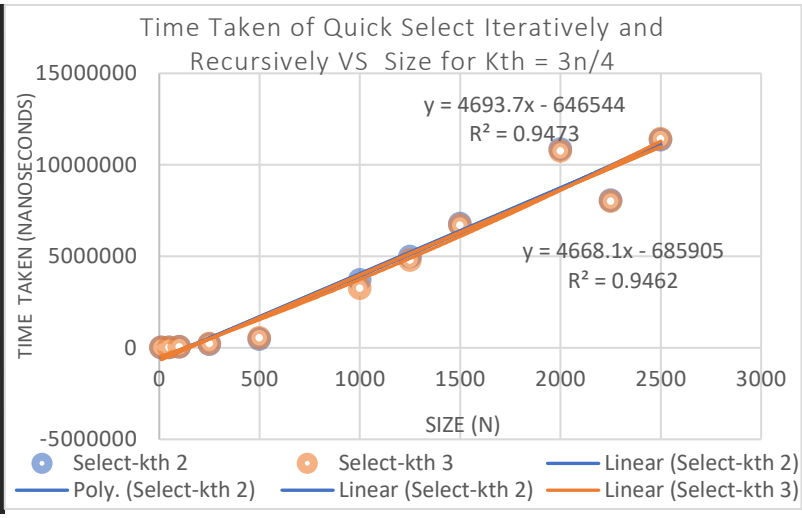




There are only a few differences between the graphs of the first test and the second test. The data points from the second test when surpassing the size of 1000 indicates the runtime of two algorithms keep increasing significantly. However, the runtime at size 2500 from the first test declined, so the second test provides more accurate result. Also, these two graphs show that the Select-kth 2 is faster than Select-kth 3 when kth = n/2.

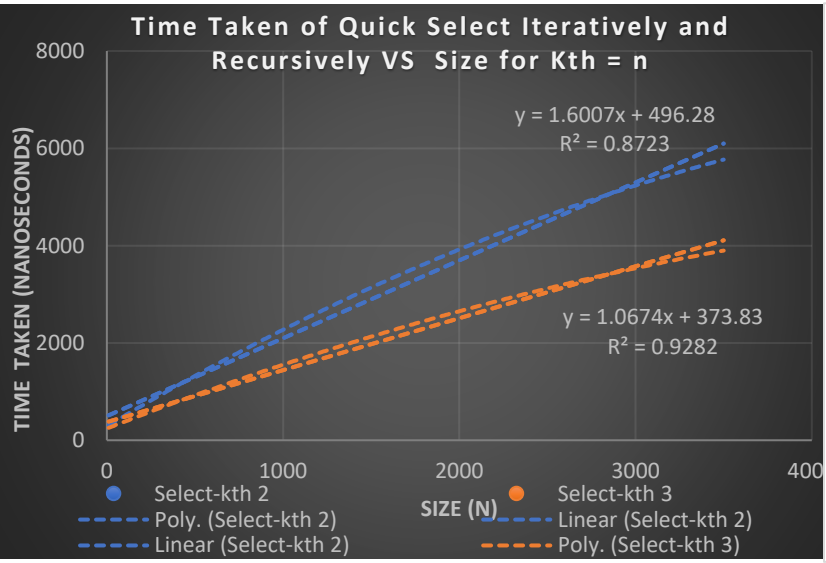
The graph shows the size versus the total time of algorithm Select-kth 2 and Select-kth 3 for a given value of kth equals to 3n/4.

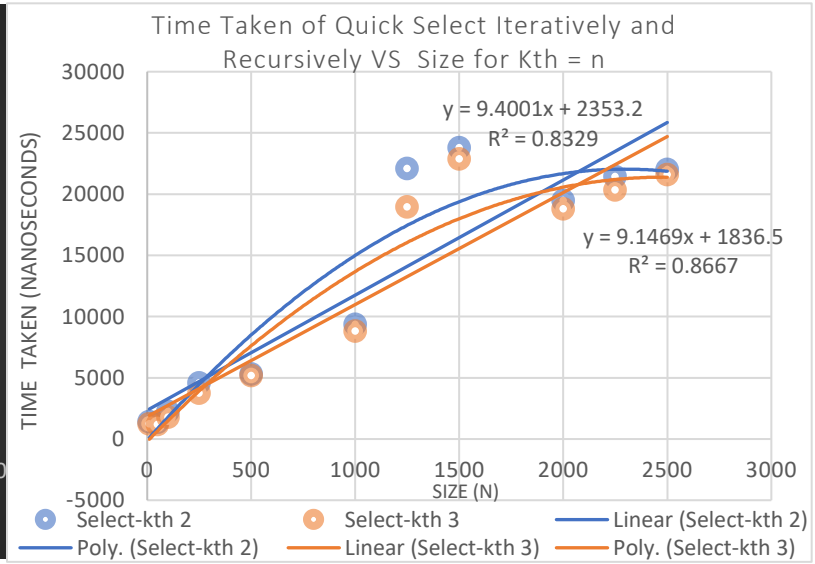




According to the graph from the first test, it seems like two algorithms have the time complexity of O(n^2) because most of data points lie on the trend line of polynomial. However, the second graph provides the accurate result where the trend line is linear, which is the best case of quick select that has the time complexity of O(n), and all the data points lie on the linear trend line perfectly. However, these two graphs indicate that when kth = 3n/4, Select-kth 3 is faster than Select-kth 2.

The graph shows the size versus the total time of algorithm Select-kth 2 and Select-kth 3 for a given value of kth equals to n.





According to the graph from the first test shows that the data points of the Select-kth 3 lie on the linear trend line perfectly, and the trend lines of polynomial and linear are about the same. Unfortunately, these two graphs show that Select-kth 2 is slower than Select-kth3, but the graph from the second test points out that the time taken of two algorithms is slightly different.

At first, I only added the trendline for linear, so I did not understand why the data that I have is not fully structured as linear line when finding different kth values. However, the data that I have follows the theoretical complexity where the quick sort partition have the best case of O(n), and its worst case is O(n^2). After I created the trendline for both linear and polynomial of n^2, these graphs show that the result of both best case and worst case according to the size of n. Also, I think the default set of size n that I have is not distributed evenly, I think it might affect the pattern of data. For this reason, I think the size of n has affected the performance of determining the kth smallest element in the list of size n.

Part 4: Select-kth 4 Versus Select-kth 1

First Experiment:

- According to the tables from the first experiment of size (n) versus the time taken to find kth in each algorithm, shows that the smaller amount of n, the Select-kth 4 would probably become faster than Select-kth 1.
- For example, the Select-kth 4 is faster than Select-kth 1 in the following scenarios:
    - Finding kth = n at size 10, 50, 100, and 250
    - Finding kth = 3n/4 at size 10
    - Finding kth = 1 at size 50

Second Experiment:

- According to the tables from the second experiment of size (n) versus the time taken to find kth in each algorithm when running the program for 40 times, shows that only a few amounts of n, the Select-kth 4 is faster than Select-kth 1.
- For example:
    - Finding kth = n at size 10 and 50
    - Finding kth = 3n/4 at size 10
    - Finding kth = 1 at size 50

According to the theoretical complexity states that medians algorithm is always faster than merge select. However, the result that I have from the first and the second experiment when running the algorithms for 20 and 40 times indicates that merge select would probably become faster. Although the result from two experiments does not correspond to the theoretical analysis, where the merge select has the time complexity of O(n log n) and the mm rule has the time complexity of O(n), the data that I have make sense to me. Since I implemented the algorithm #4, median of medians with 2 types of recursive calls (the recursive call to calculate the median of medians and the recursive call to partition an array), the recursive calls would have to create stack frames; therefore, algorithm #4 would take longer time to accomplish the task. The functionality of how I implemented the algorithm #4 and how Java deals with recursive call would make the median of medians algorithm become slower than the algorithm #1 of the merge select.

Part 5: Strength and Constraints

(4) Challenges

Challenge#1

Determine the correctness of the program.

At the design stage of algorithm number 3, I tried to run it with different kth values, and it seemed to run perfectly fine. However, when I try to run it for multiple times, it started to occur some bugs when the values of kth is greater than n/2. It printed out the random value "2147483647" which I think it caused overflow. I have seen this error when I tried to implement the algorithm 1, algorithm 2 iteratively, and algorithm 2 recursively. The error I had would be something wrong with the logic, and it would have to deal with the index of the array because when I changed the condition to determine whether the pivot position is the kth element, it worked out afterward. Eventually, I was able to manage to fix the bugs and have the program run smoothly.

Challenge#2

Declare the proper set of numbers of size n.

I had determined the wrong set of size n, I should have incremented the size of n evenly, in other words, I should have incremented by 25 or 50 every time instead of incrementing by 25 or 50 randomly. I looked at the project description and tried to understand how much I should increment the size of n, but in the project description does not specify it. As a result, I interpreted that the size of n could be any numbers as long as the size keep increasing. Basically, I hard coded and incremented the size randomly. I think if I have incremented it evenly, I would obtain more stable lines for the result on the graphs.

Conclusion:

Strength:

- Due to the line of the graph looks unstable when performing the Select-kth 4 that applies MM rule, so I decided to test with the same set of size n, where n = {10, 50, 100, 250, 500, 1000, 1250, 1500, 2000, 2250, 2500, (3000)}, but I have it run for 40 times per each set of data when performing each algorithm. For this reason, I believe that running for many numbers of times will help to obtain the accurate results.
- I was able to obtain another set of result, so it helped me to visualize all the data I have collected to make a better algorithm analysis.
- I have learned from the result that I have; for example, I know that in theoretical analysis the mm rule is more efficient than the merge select algorithm. Maybe I could develop the recursive calls to tail recursion if it is possible. I learned from weaknesses of the program, and I would like to improve the correctness of both algorithm analysis and functionality of the program itself.

Constraints:

- When testing Select-kth 1- Select-kth 4 with the size of list n = {10, 50, 100, 250, 500, 1000, 1250, 1500, 2000, 2250, 2500, 3000, 3250,3500} for 20 numbers of times. I first set the size of n = {10, 50, 100, 250, 500, 1000, 1250, 1500, 2000, 2250, 2500, 3000, 3250,3500, 4000, 4500,

5000, 5250, 5500, 6000} because I expected that my computer would be able to handle it since I only tested it with 20 number of times. However, it turned out that my computer was not able to handle many elements in a list of unsorted numbers. It took 2 hours and 47 minutes to just start executing the array of size 4000 with kth value equals to 1. Also, my computer started to generate the heat, and I could hear my computer was making a loud noise. For this reason, I decided that the size of 3500 is the maximum of n that my computer can handle.

- Since the data I obtain shows that the kth value = n seems to be the fastest when finding kth smallest element by 4 different algorithms. For this reason, I decided to generate the second test strategies to also make a comparison of the result I got from the first test strategies. For the second test, I decided to run the program for 40 number of times with the same size of n, but the maximum capacity of size n in this test is n = 3000 for kth = 1 and n/4; n = 2500 for kth value = n/2, 3n/4, and n. It took 3 hours and 48 minutes to obtain the result of the runtime. I decided to stop when I notice that my CPU fan was working hard because it made a loud noise; also, I could feel that my laptop was experiencing a high heat inside the CPU.
- I had a slower CPU to run, and my CPU has a low memory.
- The limitation of this program is that it does not support duplicate values in the list, so the program will fail to perform correctly if there exists a duplicate value in the list.

Part 6: Program Correctness

(5) Evaluation

The project was done on time, and the Select-kth 1- Select-kth 4 algorithm analysis was explained in detail, especially the major findings on the result of how each algorithm performs. I realize that there is always an opportunity to optimize my code, but I did my best on this project with good quality of time. This project was an effective tool for learning algorithm analysis.

Please refer to the reference for more information. I included the sample output as an example to evaluate the correctness of the program; also, the reference contains the tables of time taken of each algorithm versus different kth values for two test strategies.

Reference

The first test strategies:

The table shows the total time each algorithm performs to find a give kth = 1 over the size of n = {50, 1000, 1250, 1500, 2250, 2500, 3000, 3250, 3500}.

Size (n)	Select-kth 1	Select-kth 2		Select-kth 3	Select-kth 4
10	17375	6285		7355	52105
50	16810	15170		14830	109585
100	100980	11040		11520	2166725
250	237095	47490		88910	5723270
500	112475	159970		161260	45710935
1000	81645	557275		550575	272630195
1250	123580	1010275		1001685	818495970
1500	195765	1965165		1913080	1727158985
2000	179765	1271790		1303150	1323131120
2250	163285	2425300		2414790	6766384810
2500	196995	3202425		3168740	11498704000
3000	228659	3511470		3483484	15662343925
3250	236625	4903530		4872665	33789775525
3500	262249	5785034		5882710	50166715274
Average:	**153807.3571**	**1776587.071**		**1776768.143**	**8719935887**

The table shows the total time each algorithm performs to find a give kth = n/4 over the size of n = {50, 1000, 1250, 1500, 2250, 2500, 3000, 3250, 3500}.

size (n)	Select-kth 1	Select-kth 2	Select-kth 3	Select-kth 4
10	17720	1985	2230	44850
50	12195	8710	7675	80450
100	35145	23015	10330	874800
250	54010	45650	42645	8580715
500	47420	138285	124910	34204405
1000	79765	490755	487290	248091280
1250	149620	1127755	1134115	882022515
1500	181600	1356070	1452740	1218166195
2000	199440	2817290	2792015	5855998565
2250	173350	2097590	2126370	5493477200
2500	180929	2868999	2865170	11319390924
3000	270694	3951360	4003555	24695087724
3250	232764	2879915	2943175	12532713060
3500	268774	2775690	2744490	10955570764
Average:	**135959**	**1470219.214**	**1481193.571**	**5231735961**

The table shows the total time each algorithm performs to find a give kth = n/2 over the size of n = {50, 1000, 1250, 1500, 2250, 2500, 3000, 3250, 3500}.

Size (n)	Select-kth 1	Select-kth 2	Select-kth 3	Select-kth 4
10	15255	1340	1590	20410
50	19415	26570	3035	91275
100	27235	7685	7750	735565
250	74205	38950	36895	5553320
500	46125	119185	118620	28154240
1000	91900	447400	441825	249482705
1250	169925	914505	908210	724873040
1500	190685	1394555	1499255	1972883650
2000	196260	2052650	2154055	5458833480
2250	269025	1852715	1849900	6603828320
2500	240995	909764	906750	2269086075
3000	230165	3350344	3404830	25412999455
3250	244935	3103574	3126100	25344244945
3500	271934	4587419	4651475	54251778365
Average:	**149147.0714**	**1343332.571**	**1365020.714**	**8737326060**

The table shows the total time each algorithm performs to find a give kth = 3n/4 over the size of n = {50, 1000, 1250, 1500, 2250, 2500, 3000, 3250, 3500}.

Size(n)	Select-kth 1	Select-kth 2	Select-kth 3	Select-kth 4
10	16975	1180	2220	16220
50	17205	1640	1625	92345
100	26250	4755	5095	551010
250	74430	23250	21135	3539710
500	36500	61720	58105	15782830
1000	86870	251425	255260	172556750
1250	159605	290495	295045	278046300
1500	180055	847310	874240	1461139895
2000	152380	887245	901165	3068512650
2250	154965	1084125	1086130	5073182035
2500	180179	1346900	1329315	8391659939
3000	210080	1918230	1909119	19579095580
3250	245949	2375035	2313249	28685909290
3500	259385	2685070	2688279	43584121725
Average:	**128630.5714**	**841312.8571**	**838570.1429**	**7879586163**

The table shows the total time each algorithm performs to find a give kth = n over the size of n = {50, 1000, 1250, 1500, 2250, 2500, 3000, 3250, 3500}.

Size (n)	Select-kth 1	Select-kth 2	Select-kth 3	Select-kth 4
10	12090	570	510	6185
50	15755	280	230	3770
100	26225	505	355	25420
250	47240	465	380	40080
500	29370	965	670	147520
1000	75615	1630	1305	979115
1250	165180	3835	2345	3220135
1500	182435	4630	2980	5667035
2000	136235	3170	2230	9343250
2250	162565	4440	2565	14345145
2500	183615	3825	2940	21600524
3000	214715	4319	3430	41304889
3250	232760	5500	3775	54849110
3500	269165	6684	4105	76665590
Average:	**125211.7857**	**2915.571429**	**1987.142857**	**16299840.57**

The second test strategies:

The table shows the total time each algorithm performs to find a give kth = 1 over the size of n = {50, 1000, 1250, 1500, 2250, 2500, 3000}.

Size (n)	Select-kth 1	Select-kth 2	Select-kth 3	Select-kth 4
10	36190	24470	31284	234087
50	68265	70040	96759	446037
100	83030	87094	100337	3557992
250	76889	507747	850632	11092180
500	78722	1205597	1220712	54072867
1000	144949	4247984	4273157	464530735
1250	417754	12480117	12043202	2040377399
1500	387322	16041462	15971079	3856061975
2000	552254	8559037	8165130	1078931459
2250	370494	24512770	24649792	16426217920
2500	386565	26681592	26931085	22800276532
3000	425982	33660227	33812272	42080576142
	252368	**10673178.08**	**10678786.75**	**7401364610**

The table shows the total time each algorithm performs to find a give kth = n/4 over the size of n = {50, 1000, 1250, 1500, 2250, 2500, 3000}.

Size(n)	Select-kth 1	Select-kth 2	Select-kth 3	Select-kth 4
10	21182	12012	12152	32022
50	62225	22412	22535	214362
100	40669	70992	66702	587745
250	81709	321359	332990	6454552
500	70697	1034249	1048595	49421145
1000	152129	1430295	1465600	95517890
1250	342962	12678315	11743412	2095632460
1500	362620	15053657	15435234	4135838655
2000	505725	27735259	27203647	14586924919
2250	339575	20813362	20990560	15074434499
2500	383610	21314215	21252160	17365325650
3000	453597	28521057	28540235	34316382032
	234725	**10750598.67**	**10676151.83**	**7310563828**

The table shows the total time each algorithm performs to find a give kth = n/2 over the size of n = {50, 1000, 1250, 1500, 2250, 2500}.

Size(n)	Select-kth 1	Select-kth 2	Select-kth 3	Select-kth 4
10	10320	9042	9400	21599
50	102919	17570	188492	352945
100	27564	97285	69692	1090165
250	66499	263057	253497	5759660
500	68464	791910	805917	41644872
1000	145912	2192977	2165830	274514229
1250	310487	8055037	7951377	1842407549
1500	372679	11398192	11785162	3540512324
2000	403287	17881194	18458020	15226801055
2250	569849	25430910	26582225	19788703817
2500	365872	18814327	18862822	20550497832
	222168.3636	**7722863.727**	**7921130.364**	**5570209641**

The table shows the total time each algorithm performs to find a give kth = 3n/4 over the size of n = {50, 1000, 1250, 1500, 2250, 2500}.

Size (n)	Select-kth 1	Select-kth 2	Select-kth 3	Select-kth 4
10	10010	6937	6697	16489
50	60192	10969	12017	74992
100	30135	39154	52397	419277
250	54230	193822	216922	4928687
500	101452	470272	539642	28124237
1000	216935	3693647	3246540	500145857
1250	349794	4972247	4785202	1244731992
1500	406577	6782210	6693964	2720314010
2000	418427	10859945	10743462	9157609957
2250	350980	8061542	8007102	8620929355
2500	501782	11352162	11413815	15819953115
	227319.4545	**4222082.455**	**4156160**	**3463386179**

The table shows the total time each algorithm performs to find a give kth = n over the size of n = {50, 1000, 1250, 1500, 2250, 2500}.

Size (n)	Select-kth 1	Select-kth 2	Select-kth 3	Select-kth 4
10	9570	1412	1232	2834
50	57852	1362	1185	5315
100	41154	2255	1800	76019
250	70014	4577	3740	139507
500	71690	5307	5192	328760
1000	136442	9372	8824	1764120
1250	317602	22094	18967	6277352
1500	376834	23807	22887	11670819
2000	313132	19495	18809	19613215
2250	347555	21442	20335	29003130
2500	363040	22017	21597	39382872
	191353.1818	**12103.63636**	**11324.36364**	**9842176.636**

Full Sample Output:

```
Given Kth = 1, then test with the same list of numbers of size 10 for 20 times.
The given unsorted list of numbers is:
1007 5760 51 859 5502 6680 3521 9502 5980 2247
Time taken of Algorithm #1 at trial #1 is:                          28600
Time taken of Algorithm #2 iteratively at trial #1 is: 13100
Time taken of Algorithm #2 recursively at trial #1 is: 11500
Time taken of Algorithm #3 at trial #1 is:                          654100
Time taken of Algorithm #1 at trial #2 is:                          16300
Time taken of Algorithm #2 iteratively at trial #2 is: 10000
Time taken of Algorithm #2 recursively at trial #2 is: 8100
Time taken of Algorithm #3 at trial #2 is:                          35100
Time taken of Algorithm #1 at trial #3 is:                          20500
Time taken of Algorithm #2 iteratively at trial #3 is: 7400
Time taken of Algorithm #2 recursively at trial #3 is: 7700
Time taken of Algorithm #3 at trial #3 is:                          34300
Time taken of Algorithm #1 at trial #4 is:                          15300
Time taken of Algorithm #2 iteratively at trial #4 is: 7500
Time taken of Algorithm #2 recursively at trial #4 is: 23200
Time taken of Algorithm #3 at trial #4 is:                          41200
Time taken of Algorithm #1 at trial #5 is:                          24200
Time taken of Algorithm #2 iteratively at trial #5 is: 13400
Time taken of Algorithm #2 recursively at trial #5 is: 8900
Time taken of Algorithm #3 at trial #5 is:                          32200
Time taken of Algorithm #1 at trial #6 is:                          14800
Time taken of Algorithm #2 iteratively at trial #6 is: 7200
Time taken of Algorithm #2 recursively at trial #6 is: 13200
Time taken of Algorithm #3 at trial #6 is:                          35700
Time taken of Algorithm #1 at trial #7 is:                          15600
Time taken of Algorithm #2 iteratively at trial #7 is: 11500
Time taken of Algorithm #2 recursively at trial #7 is: 8100
Time taken of Algorithm #3 at trial #7 is:                          41100
Time taken of Algorithm #1 at trial #8 is:                          20600
Time taken of Algorithm #2 iteratively at trial #8 is: 5200
Time taken of Algorithm #2 recursively at trial #8 is: 5400
Time taken of Algorithm #3 at trial #8 is:                          30900
Time taken of Algorithm #1 at trial #9 is:                          16200
Time taken of Algorithm #2 iteratively at trial #9 is: 6200
Time taken of Algorithm #2 recursively at trial #9 is: 7400
Time taken of Algorithm #3 at trial #9 is:                          33400
Time taken of Algorithm #1 at trial #10 is:                         22700
Time taken of Algorithm #2 iteratively at trial #10 is: 77700
Time taken of Algorithm #2 recursively at trial #10 is: 6300
Time taken of Algorithm #3 at trial #10 is:                         35900
Time taken of Algorithm #1 at trial #11 is:                         21400
Time taken of Algorithm #2 iteratively at trial #11 is: 5600
Time taken of Algorithm #2 recursively at trial #11 is: 9500
Time taken of Algorithm #3 at trial #11 is:                         34900
Time taken of Algorithm #1 at trial #12 is:                         16100
Time taken of Algorithm #2 iteratively at trial #12 is: 5900
Time taken of Algorithm #2 recursively at trial #12 is: 6100
Time taken of Algorithm #3 at trial #12 is:                         33200
Time taken of Algorithm #1 at trial #13 is:                         22100
Time taken of Algorithm #2 iteratively at trial #13 is: 2600
Time taken of Algorithm #2 recursively at trial #13 is: 2800
Time taken of Algorithm #3 at trial #13 is:                         34400
Time taken of Algorithm #1 at trial #14 is:                         21600
Time taken of Algorithm #2 iteratively at trial #14 is: 3400
Time taken of Algorithm #2 recursively at trial #14 is: 4600
Time taken of Algorithm #3 at trial #14 is:                         30800
Time taken of Algorithm #1 at trial #15 is:                         15200
Time taken of Algorithm #2 iteratively at trial #15 is: 2100
Time taken of Algorithm #2 recursively at trial #15 is: 2600
Time taken of Algorithm #3 at trial #15 is:                         44100
Time taken of Algorithm #1 at trial #16 is:                         21400
Time taken of Algorithm #2 iteratively at trial #16 is: 2200
Time taken of Algorithm #2 recursively at trial #16 is: 2800
Time taken of Algorithm #3 at trial #16 is:                         30200
Time taken of Algorithm #1 at trial #17 is:                         16200
```

```
Time taken of Algorithm #2 iteratively at trial #17 is: 2300
Time taken of Algorithm #2 recursively at trial #17 is: 2500
Time taken of Algorithm #3 at trial #17 is:                          27600
Time taken of Algorithm #1 at trial #18 is:                          24800
Time taken of Algorithm #2 iteratively at trial #18 is: 2100
Time taken of Algorithm #2 recursively at trial #18 is: 2600
Time taken of Algorithm #3 at trial #18 is:                          29200
Time taken of Algorithm #1 at trial #19 is:                          7500
Time taken of Algorithm #2 iteratively at trial #19 is: 2300
Time taken of Algorithm #2 recursively at trial #19 is: 2800
Time taken of Algorithm #3 at trial #19 is:                          39100
Time taken of Algorithm #1 at trial #20 is:                          7200
Time taken of Algorithm #2 iteratively at trial #20 is: 2600
Time taken of Algorithm #2 recursively at trial #20 is: 2900
Time taken of Algorithm #3 at trial #20 is:                          24000
Kth smallest element of each algorithm is:
Algorithm #1:                              51
Algorithm #2 Iteratively:         51
Algorithm #2 Recursively:         51
Algorithm #3:                              51
The given sorted list of numbers is:
51 859 1007 2247 3521 5502 5760 5980 6680 9502
Average time of Algorithm #1:                                  18415
Average time of Algorithm #2 Iteratively:  9515
Average time of Algorithm #2 Recursively:  6950
Average time of Algorithm #3:                                  65070
```

```
Given Kth = n/4, then test with the same list of numbers of size 10 for 20 times.
The given unsorted list of numbers is:
4993 6108 9196 3102 4801 998 5327 5400 2495 2423
Time taken of Algorithm #1 at trial #1 is:                           21900
Time taken of Algorithm #2 iteratively at trial #1 is: 2000
Time taken of Algorithm #2 recursively at trial #1 is: 2300
Time taken of Algorithm #3 at trial #1 is:                           45000
Time taken of Algorithm #1 at trial #2 is:                           12400
Time taken of Algorithm #2 iteratively at trial #2 is: 1700
Time taken of Algorithm #2 recursively at trial #2 is: 2400
Time taken of Algorithm #3 at trial #2 is:                           6900
Time taken of Algorithm #1 at trial #3 is:                           7000
Time taken of Algorithm #2 iteratively at trial #3 is: 2200
Time taken of Algorithm #2 recursively at trial #3 is: 3200
Time taken of Algorithm #3 at trial #3 is:                           9800
Time taken of Algorithm #1 at trial #4 is:                           25100
Time taken of Algorithm #2 iteratively at trial #4 is: 2100
Time taken of Algorithm #2 recursively at trial #4 is: 5200
Time taken of Algorithm #3 at trial #4 is:                           5800
Time taken of Algorithm #1 at trial #5 is:                           6900
Time taken of Algorithm #2 iteratively at trial #5 is: 2300
Time taken of Algorithm #2 recursively at trial #5 is: 2600
Time taken of Algorithm #3 at trial #5 is:                           22200
Time taken of Algorithm #1 at trial #6 is:                           18900
Time taken of Algorithm #2 iteratively at trial #6 is: 4500
Time taken of Algorithm #2 recursively at trial #6 is: 6200
Time taken of Algorithm #3 at trial #6 is:                           5200
Time taken of Algorithm #1 at trial #7 is:                           11400
Time taken of Algorithm #2 iteratively at trial #7 is: 2300
Time taken of Algorithm #2 recursively at trial #7 is: 5200
Time taken of Algorithm #3 at trial #7 is:                           4200
Time taken of Algorithm #1 at trial #8 is:                           7000
Time taken of Algorithm #2 iteratively at trial #8 is: 1900
Time taken of Algorithm #2 recursively at trial #8 is: 2600
Time taken of Algorithm #3 at trial #8 is:                           4100
Time taken of Algorithm #1 at trial #9 is:                           13100
Time taken of Algorithm #2 iteratively at trial #9 is: 2400
Time taken of Algorithm #2 recursively at trial #9 is: 2600
Time taken of Algorithm #3 at trial #9 is:                           4200
Time taken of Algorithm #1 at trial #10 is:                          6900
Time taken of Algorithm #2 iteratively at trial #10 is: 3200
Time taken of Algorithm #2 recursively at trial #10 is: 6800
Time taken of Algorithm #3 at trial #10 is:                          5600
```

```
Time taken of Algorithm #1 at trial #11 is:                          18300
Time taken of Algorithm #2 iteratively at trial #11 is: 4800
Time taken of Algorithm #2 recursively at trial #11 is: 4200
Time taken of Algorithm #3 at trial #11 is:                          4900
Time taken of Algorithm #1 at trial #12 is:                          17800
Time taken of Algorithm #2 iteratively at trial #12 is: 4700
Time taken of Algorithm #2 recursively at trial #12 is: 5300
Time taken of Algorithm #3 at trial #12 is:                          6600
Time taken of Algorithm #1 at trial #13 is:                          18000
Time taken of Algorithm #2 iteratively at trial #13 is: 5200
Time taken of Algorithm #2 recursively at trial #13 is: 2800
Time taken of Algorithm #3 at trial #13 is:                          18700
Time taken of Algorithm #1 at trial #14 is:                          24200
Time taken of Algorithm #2 iteratively at trial #14 is: 2300
Time taken of Algorithm #2 recursively at trial #14 is: 4300
Time taken of Algorithm #3 at trial #14 is:                          4900
Time taken of Algorithm #1 at trial #15 is:                          13900
Time taken of Algorithm #2 iteratively at trial #15 is: 2000
Time taken of Algorithm #2 recursively at trial #15 is: 2800
Time taken of Algorithm #3 at trial #15 is:                          6300
Time taken of Algorithm #1 at trial #16 is:                          13000
Time taken of Algorithm #2 iteratively at trial #16 is: 2300
Time taken of Algorithm #2 recursively at trial #16 is: 2900
Time taken of Algorithm #3 at trial #16 is:                          5200
Time taken of Algorithm #1 at trial #17 is:                          7100
Time taken of Algorithm #2 iteratively at trial #17 is: 2100
Time taken of Algorithm #2 recursively at trial #17 is: 2700
Time taken of Algorithm #3 at trial #17 is:                          9700
Time taken of Algorithm #1 at trial #18 is:                          7000
Time taken of Algorithm #2 iteratively at trial #18 is: 5700
Time taken of Algorithm #2 recursively at trial #18 is: 3000
Time taken of Algorithm #3 at trial #18 is:                          5800
Time taken of Algorithm #1 at trial #19 is:                          9500
Time taken of Algorithm #2 iteratively at trial #19 is: 1700
Time taken of Algorithm #2 recursively at trial #19 is: 2500
Time taken of Algorithm #3 at trial #19 is:                          4800
Time taken of Algorithm #1 at trial #20 is:                          7200
Time taken of Algorithm #2 iteratively at trial #20 is: 2200
Time taken of Algorithm #2 recursively at trial #20 is: 2500
Time taken of Algorithm #3 at trial #20 is:                          4600
Kth smallest element of each algorithm is:
Algorithm #1:                               2423
Algorithm #2 Iteratively:         2423
Algorithm #2 Recursively:         2423
Algorithm #3:                               2423
The given sorted list of numbers is:
998 2423 2495 3102 4801 4993 5327 5400 6108 9196
Average time of Algorithm #1:                                  13330
Average time of Algorithm #2 Iteratively:  2880
Average time of Algorithm #2 Recursively:  3605
Average time of Algorithm #3:                                  9225
```

```
Given Kth = n/2, then test with the same list of numbers of size 10 for 20 times.
The given unsorted list of numbers is:
5517 9023 4738 1470 350 7495 4792 1558 9169 3490

Time taken of Algorithm #1 at trial #1 is:                           13800
Time taken of Algorithm #2 iteratively at trial #1 is: 2700
Time taken of Algorithm #2 recursively at trial #1 is: 2000
Time taken of Algorithm #3 at trial #1 is:                           7000
Time taken of Algorithm #1 at trial #2 is:                           8300
Time taken of Algorithm #2 iteratively at trial #2 is: 2000
Time taken of Algorithm #2 recursively at trial #2 is: 2200
Time taken of Algorithm #3 at trial #2 is:                           4200
Time taken of Algorithm #1 at trial #3 is:                           15400
Time taken of Algorithm #2 iteratively at trial #3 is: 2000
Time taken of Algorithm #2 recursively at trial #3 is: 2000
Time taken of Algorithm #3 at trial #3 is:                           3900
Time taken of Algorithm #1 at trial #4 is:                           12200
Time taken of Algorithm #2 iteratively at trial #4 is: 2000
```

```
Time taken of Algorithm #2 recursively at trial #4 is: 2400
Time taken of Algorithm #3 at trial #4 is:                          9300
Time taken of Algorithm #1 at trial #5 is:                          7800
Time taken of Algorithm #2 iteratively at trial #5 is: 1800
Time taken of Algorithm #2 recursively at trial #5 is: 2100
Time taken of Algorithm #3 at trial #5 is:                          3900
Time taken of Algorithm #1 at trial #6 is:                          6900
Time taken of Algorithm #2 iteratively at trial #6 is: 2000
Time taken of Algorithm #2 recursively at trial #6 is: 2100
Time taken of Algorithm #3 at trial #6 is:                          3700
Time taken of Algorithm #1 at trial #7 is:                          6900
Time taken of Algorithm #2 iteratively at trial #7 is: 1900
Time taken of Algorithm #2 recursively at trial #7 is: 2200
Time taken of Algorithm #3 at trial #7 is:                          3500
Time taken of Algorithm #1 at trial #8 is:                          11700
Time taken of Algorithm #2 iteratively at trial #8 is: 1900
Time taken of Algorithm #2 recursively at trial #8 is: 2200
Time taken of Algorithm #3 at trial #8 is:                          3100
Time taken of Algorithm #1 at trial #9 is:                          7400
Time taken of Algorithm #2 iteratively at trial #9 is: 1800
Time taken of Algorithm #2 recursively at trial #9 is: 2100
Time taken of Algorithm #3 at trial #9 is:                          3500
Time taken of Algorithm #1 at trial #10 is:                         17900
Time taken of Algorithm #2 iteratively at trial #10 is: 1800
Time taken of Algorithm #2 recursively at trial #10 is: 2200
Time taken of Algorithm #3 at trial #10 is:                         4700
Time taken of Algorithm #1 at trial #11 is:                         7100
Time taken of Algorithm #2 iteratively at trial #11 is: 1900
Time taken of Algorithm #2 recursively at trial #11 is: 2000
Time taken of Algorithm #3 at trial #11 is:                         4000
Time taken of Algorithm #1 at trial #12 is:                         12200
Time taken of Algorithm #2 iteratively at trial #12 is: 1900
Time taken of Algorithm #2 recursively at trial #12 is: 2200
Time taken of Algorithm #3 at trial #12 is:                         3600
Time taken of Algorithm #1 at trial #13 is:                         8200
Time taken of Algorithm #2 iteratively at trial #13 is: 2300
Time taken of Algorithm #2 recursively at trial #13 is: 2300
Time taken of Algorithm #3 at trial #13 is:                         9000
Time taken of Algorithm #1 at trial #14 is:                         6800
Time taken of Algorithm #2 iteratively at trial #14 is: 2100
Time taken of Algorithm #2 recursively at trial #14 is: 2300
Time taken of Algorithm #3 at trial #14 is:                         4000
Time taken of Algorithm #1 at trial #15 is:                         7100
Time taken of Algorithm #2 iteratively at trial #15 is: 1800
Time taken of Algorithm #2 recursively at trial #15 is: 2300
Time taken of Algorithm #3 at trial #15 is:                         6400
Time taken of Algorithm #1 at trial #16 is:                         7000
Time taken of Algorithm #2 iteratively at trial #16 is: 1800
Time taken of Algorithm #2 recursively at trial #16 is: 2200
Time taken of Algorithm #3 at trial #16 is:                         5300
Time taken of Algorithm #1 at trial #17 is:                         21200
Time taken of Algorithm #2 iteratively at trial #17 is: 1900
Time taken of Algorithm #2 recursively at trial #17 is: 2500
Time taken of Algorithm #3 at trial #17 is:                         4000
Time taken of Algorithm #1 at trial #18 is:                         7100
Time taken of Algorithm #2 iteratively at trial #18 is: 1600
Time taken of Algorithm #2 recursively at trial #18 is: 2000
Time taken of Algorithm #3 at trial #18 is:                         3800
Time taken of Algorithm #1 at trial #19 is:                         12100
Time taken of Algorithm #2 iteratively at trial #19 is: 1900
Time taken of Algorithm #2 recursively at trial #19 is: 2400
Time taken of Algorithm #3 at trial #19 is:                         4000
Time taken of Algorithm #1 at trial #20 is:                         7700
Time taken of Algorithm #2 iteratively at trial #20 is: 1800
Time taken of Algorithm #2 recursively at trial #20 is: 2300
Time taken of Algorithm #3 at trial #20 is:                         4000
Kth smallest element of each algorithm is:
Algorithm #1:                               4738
Algorithm #2 Iteratively:         4738
Algorithm #2 Recursively:         4738
```

```
Algorithm #3:                              4738
The given sorted list of numbers is:
350 1470 1558 3490 4738 4792 5517 7495 9023 9169
Average time of Algorithm #1:                                10240
Average time of Algorithm #2 Iteratively:  1945
Average time of Algorithm #2 Recursively:  2200
Average time of Algorithm #3:                                4745
```

```
Given Kth = 3*n/4, then test with the same list of numbers of size 10 for 20 times.
The given unsorted list of numbers is:
3485 3364 2827 1563 3004 7319 7887 9589 7171 511
Time taken of Algorithm #1 at trial #1 is:                           8800
Time taken of Algorithm #2 iteratively at trial #1 is: 2300
Time taken of Algorithm #2 recursively at trial #1 is: 2300
Time taken of Algorithm #3 at trial #1 is:                           4500
Time taken of Algorithm #1 at trial #2 is:                           2700
Time taken of Algorithm #2 iteratively at trial #2 is: 1700
Time taken of Algorithm #2 recursively at trial #2 is: 2100
Time taken of Algorithm #3 at trial #2 is:                           8300
Time taken of Algorithm #1 at trial #3 is:                           2600
Time taken of Algorithm #2 iteratively at trial #3 is: 1700
Time taken of Algorithm #2 recursively at trial #3 is: 2200
Time taken of Algorithm #3 at trial #3 is:                           3400
Time taken of Algorithm #1 at trial #4 is:                           2500
Time taken of Algorithm #2 iteratively at trial #4 is: 1500
Time taken of Algorithm #2 recursively at trial #4 is: 2000
Time taken of Algorithm #3 at trial #4 is:                           3100
Time taken of Algorithm #1 at trial #5 is:                           2400
Time taken of Algorithm #2 iteratively at trial #5 is: 1600
Time taken of Algorithm #2 recursively at trial #5 is: 1900
Time taken of Algorithm #3 at trial #5 is:                           2900
Time taken of Algorithm #1 at trial #6 is:                           8500
Time taken of Algorithm #2 iteratively at trial #6 is: 2200
Time taken of Algorithm #2 recursively at trial #6 is: 2900
Time taken of Algorithm #3 at trial #6 is:                           2800
Time taken of Algorithm #1 at trial #7 is:                           2100
Time taken of Algorithm #2 iteratively at trial #7 is: 1500
Time taken of Algorithm #2 recursively at trial #7 is: 1800
Time taken of Algorithm #3 at trial #7 is:                           3800
Time taken of Algorithm #1 at trial #8 is:                           6900
Time taken of Algorithm #2 iteratively at trial #8 is: 1600
Time taken of Algorithm #2 recursively at trial #8 is: 2300
Time taken of Algorithm #3 at trial #8 is:                           3300
Time taken of Algorithm #1 at trial #9 is:                           2300
Time taken of Algorithm #2 iteratively at trial #9 is: 1600
Time taken of Algorithm #2 recursively at trial #9 is: 2000
Time taken of Algorithm #3 at trial #9 is:                           3200
Time taken of Algorithm #1 at trial #10 is:                          6800
Time taken of Algorithm #2 iteratively at trial #10 is: 1600
Time taken of Algorithm #2 recursively at trial #10 is: 1800
Time taken of Algorithm #3 at trial #10 is:                          3000
Time taken of Algorithm #1 at trial #11 is:                          2200
Time taken of Algorithm #2 iteratively at trial #11 is: 1600
Time taken of Algorithm #2 recursively at trial #11 is: 1900
Time taken of Algorithm #3 at trial #11 is:                          3100
Time taken of Algorithm #1 at trial #12 is:                          6200
Time taken of Algorithm #2 iteratively at trial #12 is: 2400
Time taken of Algorithm #2 recursively at trial #12 is: 2000
Time taken of Algorithm #3 at trial #12 is:                          3200
Time taken of Algorithm #1 at trial #13 is:                          2300
Time taken of Algorithm #2 iteratively at trial #13 is: 1700
Time taken of Algorithm #2 recursively at trial #13 is: 2100
Time taken of Algorithm #3 at trial #13 is:                          3100
Time taken of Algorithm #1 at trial #14 is:                          6700
Time taken of Algorithm #2 iteratively at trial #14 is: 1600
Time taken of Algorithm #2 recursively at trial #14 is: 2000
Time taken of Algorithm #3 at trial #14 is:                          2900
Time taken of Algorithm #1 at trial #15 is:                          2100
Time taken of Algorithm #2 iteratively at trial #15 is: 1700
Time taken of Algorithm #2 recursively at trial #15 is: 1600
```

```
Time taken of Algorithm #3 at trial #15 is:                          7300
Time taken of Algorithm #1 at trial #16 is:                          2000
Time taken of Algorithm #2 iteratively at trial #16 is: 1600
Time taken of Algorithm #2 recursively at trial #16 is: 2000
Time taken of Algorithm #3 at trial #16 is:                          3000
Time taken of Algorithm #1 at trial #17 is:                          2100
Time taken of Algorithm #2 iteratively at trial #17 is: 2200
Time taken of Algorithm #2 recursively at trial #17 is: 2600
Time taken of Algorithm #3 at trial #17 is:                          7200
Time taken of Algorithm #1 at trial #18 is:                          2100
Time taken of Algorithm #2 iteratively at trial #18 is: 1800
Time taken of Algorithm #2 recursively at trial #18 is: 1800
Time taken of Algorithm #3 at trial #18 is:                          7900
Time taken of Algorithm #1 at trial #19 is:                          6100
Time taken of Algorithm #2 iteratively at trial #19 is: 1900
Time taken of Algorithm #2 recursively at trial #19 is: 12100
Time taken of Algorithm #3 at trial #19 is:                          2900
Time taken of Algorithm #1 at trial #20 is:                          3200
Time taken of Algorithm #2 iteratively at trial #20 is: 1600
Time taken of Algorithm #2 recursively at trial #20 is: 900
Time taken of Algorithm #3 at trial #20 is:                          3700
Kth smallest element of each algorithm is:
Algorithm #1:                               7171
Algorithm #2 Iteratively:         7171
Algorithm #2 Recursively:         7171
Algorithm #3:                               7171
The given sorted list of numbers is:
511 1563 2827 3004 3364 3485 7171 7319 7887 9589
Average time of Algorithm #1:                                  4030
Average time of Algorithm #2 Iteratively:  1770
Average time of Algorithm #2 Recursively:  2515
Average time of Algorithm #3:                                  4130
```

```
Given Kth = n, then test with the same list of numbers of size 10 for 20 times.
The given unsorted list of numbers is:
8730 3303 6956 6243 2248 3879 5565 2687 8092 1090
Time taken of Algorithm #1 at trial #1 is:                           7800
Time taken of Algorithm #2 iteratively at trial #1 is: 2200
Time taken of Algorithm #2 recursively at trial #1 is: 1000
Time taken of Algorithm #3 at trial #1 is:                           3000
Time taken of Algorithm #1 at trial #2 is:                           4900
Time taken of Algorithm #2 iteratively at trial #2 is: 1900
Time taken of Algorithm #2 recursively at trial #2 is: 900
Time taken of Algorithm #3 at trial #2 is:                           3500
Time taken of Algorithm #1 at trial #3 is:                           4100
Time taken of Algorithm #2 iteratively at trial #3 is: 1500
Time taken of Algorithm #2 recursively at trial #3 is: 900
Time taken of Algorithm #3 at trial #3 is:                           1900
Time taken of Algorithm #1 at trial #4 is:                           3200
Time taken of Algorithm #2 iteratively at trial #4 is: 1400
Time taken of Algorithm #2 recursively at trial #4 is: 900
Time taken of Algorithm #3 at trial #4 is:                           2100
Time taken of Algorithm #1 at trial #5 is:                           3100
Time taken of Algorithm #2 iteratively at trial #5 is: 1400
Time taken of Algorithm #2 recursively at trial #5 is: 800
Time taken of Algorithm #3 at trial #5 is:                           1800
Time taken of Algorithm #1 at trial #6 is:                           2500
Time taken of Algorithm #2 iteratively at trial #6 is: 1900
Time taken of Algorithm #2 recursively at trial #6 is: 800
Time taken of Algorithm #3 at trial #6 is:                           2900
Time taken of Algorithm #1 at trial #7 is:                           4000
Time taken of Algorithm #2 iteratively at trial #7 is: 2500
Time taken of Algorithm #2 recursively at trial #7 is: 1100
Time taken of Algorithm #3 at trial #7 is:                           2700
Time taken of Algorithm #1 at trial #8 is:                           3900
Time taken of Algorithm #2 iteratively at trial #8 is: 2000
Time taken of Algorithm #2 recursively at trial #8 is: 800
Time taken of Algorithm #3 at trial #8 is:                           2800
Time taken of Algorithm #1 at trial #9 is:                           4400
Time taken of Algorithm #2 iteratively at trial #9 is: 1900
```

```
Time taken of Algorithm #2 recursively at trial #9 is: 900
Time taken of Algorithm #3 at trial #9 is:                          7100
Time taken of Algorithm #1 at trial #10 is:                         3500
Time taken of Algorithm #2 iteratively at trial #10 is: 1800
Time taken of Algorithm #2 recursively at trial #10 is: 700
Time taken of Algorithm #3 at trial #10 is:                         1700
Time taken of Algorithm #1 at trial #11 is:                         4400
Time taken of Algorithm #2 iteratively at trial #11 is: 1300
Time taken of Algorithm #2 recursively at trial #11 is: 1000
Time taken of Algorithm #3 at trial #11 is:                         2100
Time taken of Algorithm #1 at trial #12 is:                         4300
Time taken of Algorithm #2 iteratively at trial #12 is: 1500
Time taken of Algorithm #2 recursively at trial #12 is: 800
Time taken of Algorithm #3 at trial #12 is:                         1800
Time taken of Algorithm #1 at trial #13 is:                         3300
Time taken of Algorithm #2 iteratively at trial #13 is: 1300
Time taken of Algorithm #2 recursively at trial #13 is: 1200
Time taken of Algorithm #3 at trial #13 is:                         1700
Time taken of Algorithm #1 at trial #14 is:                         8500
Time taken of Algorithm #2 iteratively at trial #14 is: 1300
Time taken of Algorithm #2 recursively at trial #14 is: 800
Time taken of Algorithm #3 at trial #14 is:                         1800
Time taken of Algorithm #1 at trial #15 is:                         2700
Time taken of Algorithm #2 iteratively at trial #15 is: 1300
Time taken of Algorithm #2 recursively at trial #15 is: 800
Time taken of Algorithm #3 at trial #15 is:                         2000
Time taken of Algorithm #1 at trial #16 is:                         7500
Time taken of Algorithm #2 iteratively at trial #16 is: 1700
Time taken of Algorithm #2 recursively at trial #16 is: 800
Time taken of Algorithm #3 at trial #16 is:                         5000
Time taken of Algorithm #1 at trial #17 is:                         3800
Time taken of Algorithm #2 iteratively at trial #17 is: 1500
Time taken of Algorithm #2 recursively at trial #17 is: 1200
Time taken of Algorithm #3 at trial #17 is:                         2300
Time taken of Algorithm #1 at trial #18 is:                         8700
Time taken of Algorithm #2 iteratively at trial #18 is: 1400
Time taken of Algorithm #2 recursively at trial #18 is: 1100
Time taken of Algorithm #3 at trial #18 is:                         1800
Time taken of Algorithm #1 at trial #19 is:                         2900
Time taken of Algorithm #2 iteratively at trial #19 is: 1100
Time taken of Algorithm #2 recursively at trial #19 is: 800
Time taken of Algorithm #3 at trial #19 is:                         1900
Time taken of Algorithm #1 at trial #20 is:                         8000
Time taken of Algorithm #2 iteratively at trial #20 is: 1200
Time taken of Algorithm #2 recursively at trial #20 is: 800
Time taken of Algorithm #3 at trial #20 is:                         1600
Kth smallest element of each algorithm is:
Algorithm #1:                               8730
Algorithm #2 Iteratively:         8730
Algorithm #2 Recursively:         8730
Algorithm #3:                               8730
The given sorted list of numbers is:
1090 2248 2687 3303 3879 5565 6243 6956 8092 8730
Average time of Algorithm #1:                                  4775
Average time of Algorithm #2 Iteratively:  1605
Average time of Algorithm #2 Recursively:  905
Average time of Algorithm #3:                                  2575
```